ISPTEC

Instituto Superior Politécnico de Tecnologias e Ciências

Programação II

Nome: Fábio Tales Victorino

Numero: 20200921

Professor: Sediangani Sofrimento

A justificação da escolha das classes, herança e polimorfismo.

Seguindo as boas práticas da POO que nos orienta a tornar as nossas classes o mais especificos possivel.

A classe Nave define uma entidade por isso é uma classe que tem duas herança a SUBLUZ e FTL, visto que são tipos de nave.

A Classe Transporte é abstract e, isto deve-se ao facto de ter duas herança Pessoas e Materia que compartilham de mesmo comportamento mas com diferente formas de aplicação.

Foi bom usar polimorfismo porque dessa forma me permitiu trabalhar com dados vindo de outras partes o que acaba por tornar o codigo mais limpo e fácil de fazer manutenção.

Na classe Gerencia ela basicamente faz aplicação das Interface IGerenciar.

Onde implementa os metodos de forma a tornar sistema mais seguro evitando que modificalções fossem feitas a partir da package main.

A descrição resumida dos algoritmos mais complexos.

public static void viajar(List<Transporte> listTransporte, List<NaveEspacial> listNave): Este é o cabeçalho do método viajar. Ele é público (public), estático (static), não retorna um valor (void) e aceita duas listas como parâmetros: uma lista de objetos do tipo Transporte (listTransporte) e outra de objetos do tipo NaveEspacial (listNave).

if (listTransporte.isEmpty() || listNave.isEmpty()) { ... }: Aqui, o método verifica se as listas de transporte ou naves estão vazias. Se alguma delas estiver vazia, o método simplesmente retorna sem fazer mais nada.

for (Transporte transporte : listTransporte) { ... }: O método itera sobre cada objeto Transporte na lista listTransporte.

if (transporte.getEstado() == Estado.TRANSPORTAR) { ... }: Para cada objeto Transporte, verifica se o estado é igual a TRANSPORTAR. Se for, o estado é atualizado para FINALIZADO.

for (NaveEspacial naveEspacial : listNave) { ... }: Dentro do loop anterior, o método itera sobre cada objeto NaveEspacial na lista listNave.

if (naveEspacial.getTransporte().equals(transporte)) { ... }: Para cada objeto NaveEspacial, verifica se o transporte associado a essa nave é igual ao transporte atual. Se for, o método chama um método chamado chegar passando a lista de naves e o índice da nave encontrada.

public static void criarTransporte(): Este é o cabeçalho do método. Ele é público (public), estático (static), não retorna um valor (void) e não aceita nenhum parâmetro.

Verificação de Portos Vazios: O método verifica se a lista de portos (portoLista) está vazia. Se estiver, imprime “Não existe porto” e chama o método criarPorto().

Obtenção de Portos de Origem e Destino:

O método inicializa variáveis para controlar a entrada do usuário.

Ele exibe os IDs dos portos disponíveis usando o método mostrarPortoID().

Em um loop, solicita ao usuário que digite o ID do porto de origem e, em seguida, o ID do porto de destino.

Verifica se as opções são válidas e se não são iguais.

Adiciona os portos à lista listPorto.

Criação do Tipo de Transporte:

Se nenhum porto foi adicionado, o método imprime “erro ao criar” e retorna.

Pergunta ao usuário o tipo de transporte: 1 para Pessoa ou 2 para Material.

Com base na opção escolhida, chama os métodos addPessoa(listPorto) ou addMaterial(listPorto) para criar o transporte correspondente.

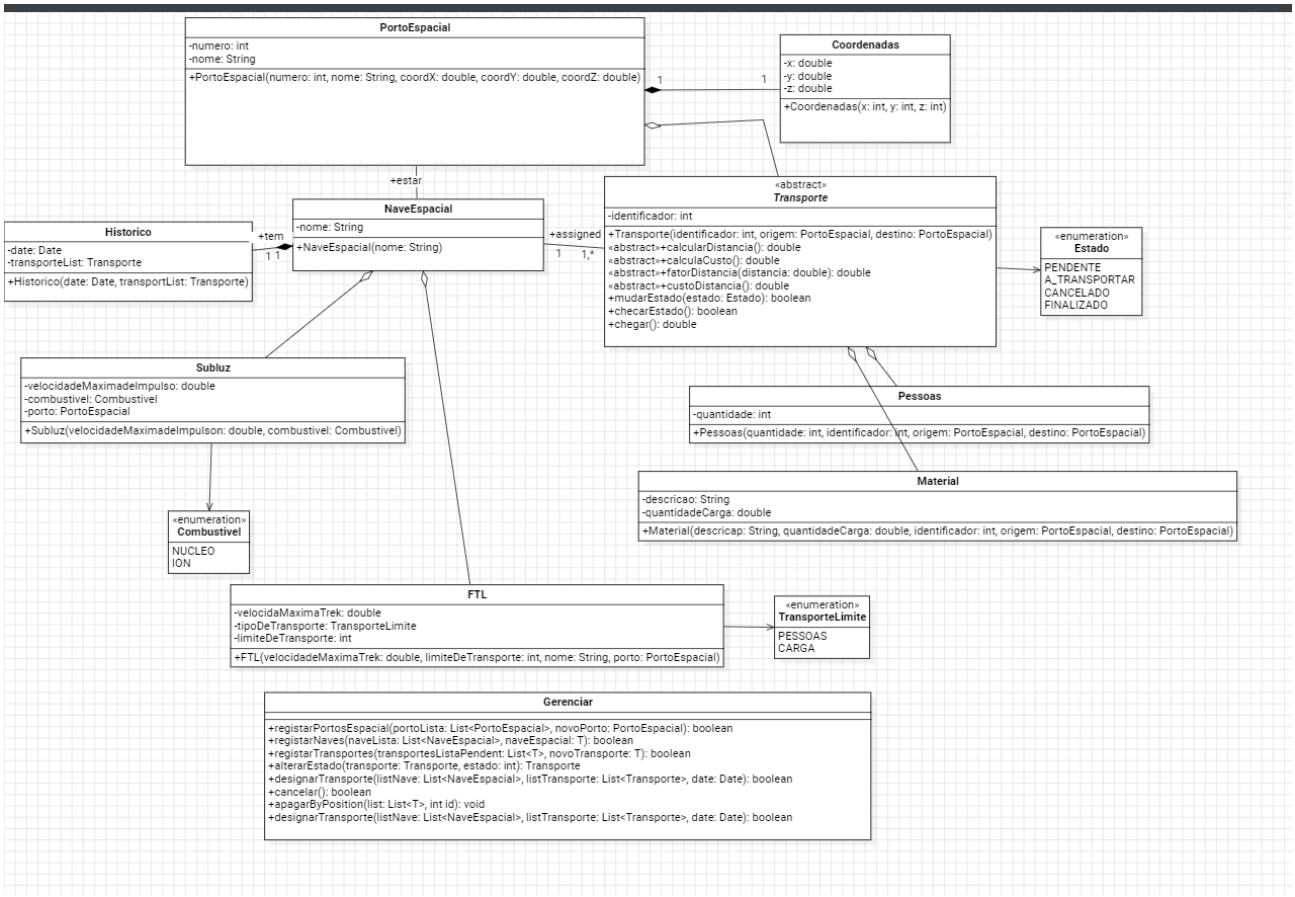
Tratamento de Exceções:

Captura exceções de entrada inválida (InputMismatchException) e imprime uma mensagem apropriada.

Também captura outras exceções genéricas e imprime uma mensagem de erro.

Organização

1. Entidades
2. Enums
3. Main


PortoEspacial
-numero: int
-nome: String
+PortoEspacial(numero: int, nome: String, coordX: double, coordY: double, coordZ: double)
Coordenadas
-x: double
-y: double
-z: double
+Coordenadas(x: int, y: int, z: int)
1
1
+estar
«abstract»
Transporte
-identificador: int
+Transporte(identificador: int, origem: PortoEspacial, destino: PortoEspacial)
«abstract»+calcularDistancia(): double
«abstract»+calculaCusto(): double
«abstract»+fatorDistancia(distancia: double): double
«abstract»+custoDistancia(): double
+mudarEstado(estado: Estado): boolean
+checarEstado(): boolean
+chegar(): double
«enumeration»
Estado
PENDENTE
A_TRANSPORTAR
CANCELADO
FINALIZADO
NaveEspacial
-nome: String
+NaveEspacial(nome: String)
Historico
-date: Date
-transporteList: Transporte
+Historico(date: Date, transportList: Transporte)
+tem
1 1
+assigned
1 1..*
Subluz
-velocidadeMaximadeImpulso: double
-combustivel: Combustivel
-porto: PortoEspacial
+Subluz(velocidadeMaximadeImpulson: double, combustivel: Combustivel)
Pessoas
-quantidade: int
+Pessoas(quantidade: int, identificador: int, origem: PortoEspacial, destino: PortoEspacial)
Material
-descricao: String
-quantidadeCarga: double
+Material(descricap: String, quantidadeCarga: double, identificador: int, origem: PortoEspacial, destino: PortoEspacial)
«enumeration»
Combustivel
NUCLEO
ION
FTL
-velocidaMaximaTrek: double
-tipoDeTransporte: TransporteLimite
-limiteDeTransporte: int
+FTL(velocidadeMaximaTrek: double, limiteDeTransporte: int, nome: String, porto: PortoEspacial)
«enumeration»
TransporteLimite
PESSOAS
CARGA
Gerenciar
+registarPortosEspacial(portoLista: List<PortoEspacial>, novoPorto: PortoEspacial): boolean
+registarNaves(naveLista: List<NaveEspacial>, naveEspacial: T): boolean
+registarTransportes(transportesListaPendent: List<T>, novoTransporte: T): boolean
+alterarEstado(transporte: Transporte, estado: int): Transporte
+designarTransporte(listNave: List<NaveEspacial>, listTransporte: List<Transporte>, date: Date): boolean
+cancelar(): boolean
+apagarByPosition(list: List<T>, int id): void
+designarTransporte(listNave: List<NaveEspacial>, listTransporte: List<Transporte>, date: Date): boolean